Isolating Order Semantics in Order-Sensitive XQuery-to-SQL Translation

Song Wang, Ling Wang, and Elke A. Rundensteiner

Worcester Polytechnic Institute, Worcester, MA 01609, USA
{songwang|lingw|rundenst}@cs.wpi.edu

Abstract. Order is essential for XML query processing. Efficient XML processing with order consideration over relational storage is non-trivial, especially for complex nested XQuery expressions. The order semantics may impede efficient query rewriting for nested query blocks. We propose a general order-sensitive XQuery processing approach involving three steps. First an algorithm is proposed for inferencing about and then isolating the order semantics in XQuery expressions specified over virtual XML views. This turns an *ordered* XQuery plan into an *unordered* one decorated with minimized *order context annotations*. Then without loss of semantics, logical optimization via XQuery rewriting can be easily applied to this transformed query plan. As last step, the translation of the optimized logical plan into SQL now correctly incorporates the order context annotations to assure the original order semantics. Our experiments illustrate the performance gains achievable by our order handling strategy.

1 Introduction

Since XQuery semantics are order sensitive, order awareness has been identified as critical for XQuery processors. Order-sensitive XML query processing has been studied for native XQuery engines, such as TIMBER [10], Natix [6] and Rainbow [1, 20]. There has also been considerable work on extending relational query engines to process XQueries over XML documents. See [5] for a survey.

Several aspects of supporting order-sensitive XQuery processing over relational storage have been successfully tackled in the literature. XML document order encoding strategies during XML loading, such as Dewey order [15], ORDPATH [9], and preorder ranks [3], have been proposed. The order-sensitive XPath to SQL translation has been studied for these different order encodings [3, 9, 15]. However, the order semantics in general impede efficient query rewriting for nested query plan blocks. We thus propose a general order-sensitive XQuery processing approach which overcomes this problem. Our solution does not rely on any specific relational order-encoding.

Our work relates to recent work on the order processing and duplication removal of matched pattern trees for the native XQuery engine Timber [10,11]. The authors propose to use hybrid collections of matched pattern trees to capture the order semantics of XQuery expressions. Although the proposed techniques are sufficient for native XQuery processing, their adoption to an XQuery engine with relational storage faces new challenges, as shown below.

R. Cooper and J. Kennedy (Eds.): BNCOD 2007, LNCS 4587, pp. 147–159, 2007.

Motivation Example 1: XML nodes matched by XQuery expressions can be heterogeneous, due to the wildcard "*" navigation step in XPath expressions or the *union* operations in the *For* and *Let* clauses. In Figure 1(a), four XML nodes ($b0$ to $b3$) match the given XPath expression for $\$b$. The intermediate result of the translated SQL thus contains four tuples[1]. We observe that every XML node in the matched pattern is represented in a column in the result table shown at the bottom of the figure. No simple sorting on any individual column can achieve the ordering of the *order by* clause, no matter what order encoding is used. Instead, we have to build an extra ordering column to record the order information corresponding to the runtime execution.

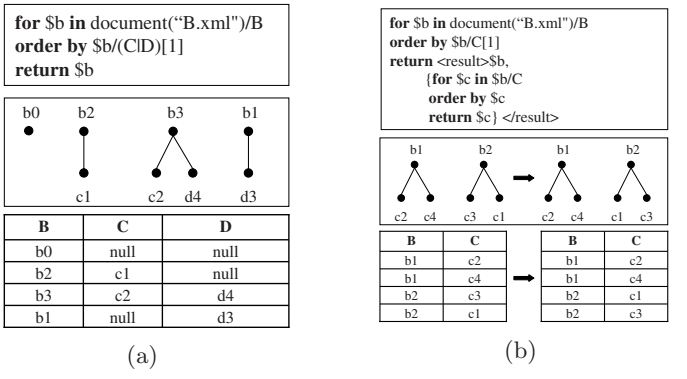

Fig. 1. Motivation Examples: (a) XQuery with Heterogeneous Matched XML Nodes; (b) Sorting in Nested XQuery Expressions

Motivation Example 2: We may need to group and sort the results of the translated SQL queries to achieve correct semantics of nested XQuery expressions. Adding simple sortings into the SQL may not be adequate. As shown in Figure 1(b), the XQuery expressions first sort the "$b" bindings by the first "C" child, and then all "C" children of each "$b" binding. The XML nodes and corresponding tuples on left show the orderings for the outer XQuery expression, while the right ones show the effect of the inner XQuery expression. Obviously we cannot achieve the correct ordering by a simple translated SQL *order by* clause.

The ordering problems shown above are unique to XQuery processing on relational XML views where order can only appear at the top-level of an SQL query. The naive approach to guarantee the correct order processing in the SQL translation is to build an extra order column for each level of the result construction. Such columns are used to capture the runtime order semantics in each of the intermediate results. Those columns can be combined and treated as Dewey order of the result XML. Then a sort at the top-level SQL query can achieve the correct ordering of the tuples. The OLAP amendment *row_number()* with *partition by* and optional *order by* clauses in SQL99 can be used to achieve this [17].

[1] For ease of illustration, b_i, c_i and d_i are used for both the XML nodes and the atomized values.